

Evolutionary Design of Secure Systems - The First Step Is Recognizing the Need for Change

Howard Lipson, Software Engineering Institute [vita³]

Copyright © 2006, 2008 Carnegie Mellon University

2006-04-19; Updated 2008-10-08

L1 / L, M, (E)⁴

In the absence of countermeasures, a system's security and survivability will degrade over time. Ensuring that you can effectively evolve the capabilities of a system in response to a changing risk environment is an essential aspect of systems design. Successful evolutionary design is dependent on the continual monitoring of the system and its environment to detect changes that can affect the risk management assumptions on which the system's security and survivability are founded.

Introduction⁵

Assumptions Evolve, and So Must Software⁶

Recognizing the Need for Evolutionary Design Activity⁷

Evolutionary Design Activities⁸

The Need for Best Practices in Evolutionary Design⁹

Introduction

A fundamental truth of system design is that, in the absence of countermeasures, a system's security and survivability will degrade over time. This degradation occurs not because any bits "rust out" or because the system shows any other manifestations of physical aging. Rather, changes in the environment or usage of a system, or changes to the elements that compose the system, often introduce new or elevated threats that the system was not designed to handle and is ill-prepared to defend itself against. Since security and survivability are system-wide properties, successfully dealing with such changes often requires revisiting every phase of the system development life cycle (SDLC) to at least some degree and poses particularly critical challenges for the assembly and integration phase.

The system you have assembled and integrated from vendor and custom components must evolve in response to a myriad of environmental changes, but the first step in evolving to meet new threats to your system's security is to *recognize* the need to modify your system—that is, to recognize changes in security and survivability risks that trigger the need to enter the evolution phase of the SDLC. The example in the next section (although not about a security failure) illustrates the critical importance of recognizing the need for evolutionary design changes.

Assumptions Evolve, and So Must Software

What has become a classic example of the catastrophic consequences that can result from a simple software error¹⁰ was the explosion of the unmanned Ariane 5 rocket during the first minute of its maiden flight on the morning of June 4, 1996 [ESA-CNES 96¹¹]. The explosion destroyed the launch vehicle's payload (a set of scientific satellites) worth on the order of \$500 million.

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/15-BSI.html (Lipson, Howard F.)

5. #dsy467-BSI_intro

6. #dsy467-BSI_assump

7. #dsy467-BSI_recog

8. #dsy467-BSI_evol

9. #dsy467-BSI_need

10. A simple arithmetic overflow doomed the Ariane 5, but taken in its operational and software engineering context, the circumstances surrounding the error were complex.

11. #dsy467-BSI_esa

The Ariane 5's flight control software reused design specifications and code from its highly successful predecessor, the Ariane 4 launch vehicle. In particular, one of the on-board modules, the Inertial Reference System, performed a data conversion of a 64-bit floating point value related to the horizontal velocity of the rocket and attempted to place the result into a 16-bit signed integer variable. This computation had never caused a problem with the Ariane 4, but the more aggressive flight path and much faster acceleration of the Ariane 5 produced a higher horizontal velocity and a corresponding data value that was too large for the 16-bit signed integer variable, causing an arithmetic overflow. A redundant backup process used the same software and failed in the same manner. The Inertial Reference System then generated some diagnostic output that was incorrectly interpreted as flight control data by other portions of the flight control system. Based on this faulty interpretation, the flight control system took actions that led to the self-destruction of the rocket.

Although arithmetic overflow is a very well-known and highly preventable error, the Ariane 4 design team did not add the exception-handling code necessary to check for arithmetic overflow and take appropriate remedial action. Based on the operating characteristics of the Ariane 4, the design team felt it was physically impossible to have a horizontal velocity large enough to cause an arithmetic overflow of a 16-bit signed integer variable. However, the reuse of this software in the Ariane 5 placed the code in a very different operating context in which the specific design assumption relating to horizontal velocity was no longer valid. Although the operating characteristics of the rocket had evolved, the underlying design assumptions based on those characteristics were not revisited by the design or testing teams, and so the software did not evolve to reflect its new operating environment. The assumptions on which the design was based no longer reflected reality.

In all there were 14 recommendations by the Flight 501 Failure Inquiry Board [ESA-CNES 96¹²]. Two of these had especially strong implications for software evolution, software architecture, code reuse, and the design of commercial off-the-shelf (COTS) based systems: recommendation 5 (first two bullet items), and recommendation 12:

R5 Review all flight software (including embedded software), and in particular:

- Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. Check these assumptions against the restrictions on use of the equipment.
- Verify the range of values taken by any internal or communication variables in the software.
- ...

R12 Give the justification documents the same attention as code. Improve the technique for keeping code and its justifications consistent.

Assumption Mismatches and the Impact of Change

Mismatches in the basic assumptions (and in particular the risk-management assumptions) on which the design of a system is based have historically been a fundamental cause of countless security, safety, and survivability problems. Architectural mismatches among components are a nearly universal source of problems [Garlan 95¹⁴] and are a direct result of mismatched assumptions. The security and survivability of COTS-based systems (and other forms of software reuse) suffer from mismatches between the assumptions made by the COTS software designers and the assumptions made by the system integrators [Lipson 02¹⁵]. Invalid assumptions made by designers about the real-world operating environment are another cause of system failures. However, even if all such assumptions were correct and were perfectly matched during the initial design, implementation, and initial deployment, the facts or circumstances on which some of these assumptions are based will invariably change over time. Any security (or other) problem arising from these changes can be considered to be a case of evolution failure—the failure of a system's designers to evolve the system in a manner that properly reflects the impact of changes (e.g., in technology, operating environment,

12. #dsy467-BSI_esa

14. #dsy467-BSI_garlan

15. #dsy467-BSI_lipson02

and business mission) on its underlying assumptions.¹⁶ Whether creating new systems and components or reusing existing ones, designing a system for evolution is a key aspect of building security in (or building quality in) from the outset.¹⁷

Recognizing the Need for Evolutionary Design Activity¹⁸

The strongest implication of the concept of evolutionary design is that the sustainment of any mission-critical system requires *perpetual design*. That is, at least to some extent, all SDLC activities must be perpetual if the quality attributes of a system are to be sustained over time. In addition to evolving a system and its components, it is also crucial for assurance cases (i.e., assurance arguments composed of artifacts and other evidence of assurance of desired system properties) to evolve as well. System characteristics that hinder or promote evolution are discussed in *Topics in Interoperability: System-of-Systems Evolution* [Carney 05¹⁹]. Some fundamental “laws” of software evolution are described in “Rules and Tools for Software Evolution Planning and Management” [Lehman 01²⁰].

It is essential that significant risk management resources should be devoted to the ongoing evolution of any mission-critical system. The successful evolutionary design of a secure and survivable system²¹ is dependent on the continual monitoring of the system and its environment to detect changes that may affect the risk management assumptions on which the security and survivability of the system are founded.

Any significant change in system requirements can certainly affect the underlying risk management assumptions, but the effects of other changes might not be as obvious. Therefore, one of the most essential uses for risk management resources would be to support security and survivability monitoring to provide early warnings of emerging threats and increased risks to the system. The amount of resources to be devoted to this activity, and to those that conduct it, will depend on executive management’s risk tolerance and their perception of the cost/benefit ratio for this effort.

Change Factors

We use the term *risk assessment triggers* to refer to the elements of a system or its environment that should be monitored, looking for changes that can affect the risk management assumptions that underlie a system’s security and survivability properties. Ideally, a best practice for system design would require that all such assumptions be explicitly specified in a design rationale document or in other system artifacts, but typically many such assumptions are merely implicit. Nevertheless, if during the lifetime of a system any of the assumptions on which its design was based no longer hold, the mission-critical properties of the system (in particular its security properties) must be reevaluated. It is therefore critical for management and the system design team to be made aware of any event or change that appears (or has the potential) to undermine one or more of those risk management assumptions. However, it is up to management and the system design team to determine whether a particular change or set of changes should trigger an evolutionary design activity and to decide on the extent of that activity.

Table 1 contains a representative set of risk assessment change factors (trigger elements) that might be tracked by an organization. Trigger events include changes in attack techniques, mission, management, staff, customers, and in the technological and legal environments. They also include changes to the elements that

16. Hence, assumption mismatches occur not only across architectures, components, and systems, they also occur over time.

17. Although this example is specifically about reuse, the lessons learned apply to the evolution of an aging component or system as well.

18. This is a revised and updated version of one of the sections contributed by Howard Lipson for “Managing Software Development for Survivable Systems” [Mead 01].

19. #dsy467-BSI_carney

20. #dsy467-BSI_lehman01

21. A survivable system is one that continues to fulfill its mission despite an attack, accident, or subsystem failure. Survivability blends security and business risk management [Lipson 99] and is based on ensuring that the quality attributes that are critical to the success of an organizational mission, such as availability and reliability, are sustained. The overall level of service may gracefully degrade under stress, but a survivable system continues to provide the essential services that support the organizational mission.

compose your system and changes to other systems with which your system is involved in a system-of-systems relationship.

Table 1. Factors that influence evolutionary design of secure systems

Change Factors (Triggers)	Examples of Trigger Events
Business and Organizational	
<i>Mission, essential services, essential quality attributes, key information resources and assets</i>	The organization's mission has changed or the system will be replaced by other organizations with different missions.
<i>Business strategies and tactics</i>	Changes in business strategies or tactics may require new types of data and processed, as well as increased connectivity among elements, imposing new security requirements.
<i>Management</i>	New executive managers may differ in their tolerance for risk and management strategies.
<i>Organizational staff</i>	Turnover may result in a lowering of staff expertise, which may reduce the ability to handle the human processes associated with security. Moreover, in a rapid change, new staff may be less trustworthy than previous staff (e.g., the lack of background checks, or there may be more remotely stationed staff).
<i>Workflow and processes</i>	Changes in organizational processes to which the system contributes may affect the overall survivability of the mission. There may be new ways of interacting with the human-machine interface.
<i>Customers</i>	New customers may be less known (and hence less trustworthy) or may require extensive access to information resources and assets, or may require a different service (e.g., higher availability) than previous customers.
<i>Collaborators</i>	A new or existing collaborator may require a deeper level of integration with business processes than your system currently supports. Or your system may become your competitor on another, requiring a more complex design.
<i>Competitors</i>	Business competitors may offer new services to your customers that your system currently cannot provide.
<i>Usage, functionality, access, or quality of service</i>	User requests for a new means of access to a system (e.g., wireless access), new ways of using an existing system, the introduction of a service, or changes in the quality of an existing service have security and survivability implications that need to be considered in any design activity undertaken in response. For example, a manufacturing plant that will now be handling a more volatile chemical ingredient requires an evolutionary redesign of the plant and safety of the plant's control systems.
Threat Environment²³	
<i>Attack techniques</i>	A new attack technique or variation has been discovered for which the system must adapt automatically or through routine maintenance (e.g., simulation for resistance, recognition, or recovery).
<i>Malicious adversaries</i>	Awareness of industrial competitors engaging in espionage (e.g., intellectual property), criminal activity, or increases in nation-state-sponsored cyberattacks may require additional system resources to be devoted to security and survivability.
Operating Environment	
<i>Technology environment</i>	Changes in the technological environment in which the system operates, such as in the systems environment, the availability of new security technologies, or technological advances in the state of the practice for the application.

	increasingly widespread dissemination of detailed knowledge of the system (and its supporting technologies) can trigger the need for improvements in system security and survivability.
<i>Physical environment</i>	The migration from wired desktops situated in a physically secure environment to laptops and other wireless mobile devices that routinely travel through hostile environments increases the possibility of both physical and logical compromise, which intensifies the need to strengthen the protection of enterprise information on those devices, as well as access to enterprise databases and applications.
Economic Environment and the Acquisition Marketplace	
<i>Cost, profit, and affordability</i>	Changing cost factors may threaten or improve a system's survivability (e.g., through changes in the cost/benefit ratio associated with various survival strategies). Affordability is a primary factor in system security and survivability. For instance, new technology could allow for an existing component at a much lower price. Greatly reduced costs could trigger an evolutionary redesign, using multiple instances (possibly from multiple vendors) to provide increased redundancy, supporting greater survivability. As another example, increased focus on short-term profits may tilt the security and survivability requirements toward higher risk, which may be reflected in cutbacks in security and maintenance contracts.
<i>Vendors and contractors</i>	A new vendor for a system component may require remote network access.
<i>COTS products</i>	You may have to replace a COTS component that is no longer supported by a new component whose contribution to system security and survivability is evaluated.
Political, Social, Legal, and Regulatory Environment	
<i>Legal environment</i>	New laws, increased enforcement of existing laws, and laws that change the equation and threaten the mission. For instance, use of the system in a jurisdiction may increase the risk of liability, which might be a concern for certain aspects of the system's security.
<i>Government regulation</i>	Changes in government regulations that mandate increased performance, competition, or quality of service may trigger the need to modify requirements to ensure that these new requirements are specified and satisfied.
<i>Certification requirements or standards</i>	Customers, regulators, and insurers may expect a system to be certified as necessary to comply with new (or changed) standards or certifications, as to reduce the actual or perceived level of risk associated with the system in a particular domain or environment. For example, business insurance policies that include cyber attack may depend on a certification of the system's security (i.e., the presentation of sufficient evidence to demonstrate that a system meets a given standard).
<i>Political and social environment</i>	Changes in privacy concerns, trust relationships, or the risk to the system may affect the security and survivability requirements that system designers must consider.
Relationships to Other Systems and Infrastructures	
<i>Dependencies and interdependencies</i>	New interconnections among the systems within an enterprise may increase the risk of failure, such as multiple systems relying on a single service or component. On a system may also be brought about by the elimination of a system.

	positions, or legacy systems, which means there is no longer fails. Increasing the interdependencies within an enterprise more likely to have pervasive effects (e.g., cascading failure in security models among the interconnected systems can require security requirements.
<i>Usage relationships</i>	Changes to systems that depend on your system (and of course that your system depends on) may require evolutionary changes to sustain the security and survivability of the overall system or
System Feedback (Lessons Learned)	
<i>System instrumentation and audits</i>	System logs allow the operations team to monitor and improve survivability of the system while it is in use (e.g., through continuous monitoring). Further analyses of this data may be used to identify survival gaps, improve security and other system quality attributes in future iterations. This analysis may also identify gaps in system instrumentation and improvements in the quality or coverage of system logs and of audits.
<i>Operational experience (attacks, accidents, and failures)</i>	Feedback from the field may lead to the discovery of new threats and survivability or may reveal existing deficiencies.
<i>Results of periodic security and survivability evaluations</i>	Troublesome results from regularly scheduled penetration testing and survivability evaluations can trigger awareness of the need for improvements.
<i>Technical society meetings, security courses, seminars, journals, news reports</i>	Awareness of lessons learned by others' system failures and improvements in your own system.

Evolutionary Design Activities

A change in one or more of the trigger elements can initiate any of a broad range of evolutionary design activities described in Table 2, from no action at all, to performing one or more system development life cycle activities, to abandonment of the system. The organizational unit responsible for monitoring for changes in risk management assumptions would initiate the consideration of an evolutionary design activity, but management and the system design team would be responsible for evaluating the impact of any trigger event that was identified and for determining the scope of any subsequent design activity in response to that event.

Table 2. Possible evolutionary design activities in response to a trigger event

Evolutionary Design Activity	Example
1. No action needed or taken	Conclude that greatly increased hiring activity does not pose a new threat to the system's mission because all new hires are subject to thorough background checks.
2. No action taken, but increase monitoring of this trigger (or set of triggers)	Increase resources devoted to monitoring feedback from the field in response to evidence from operations indicating a performance slowdown resulting from a rare combination of customer actions.
3. Further analysis needed to determine next activity, if any	Generate scenarios that reflect the discovery of a new type of cyber attack. Use these scenarios as

	input for a security analysis, the results of which may drive additional evolutionary design activities.
4. Perform a portion (delta) of one or some of the system development life cycle activities	A small change to the system architecture increases resistance to a new attack scenario.
5. Perform a portion (delta) of each of the full set of life-cycle activities	A modification to the mission touches all life-cycle activities to one extent or another.
6. Do a full redesign	A major change in the technology of the application domain, coupled with sweeping improvements in defensive technology, cannot be incorporated by evolutionary design activities alone.
7. Abandon the system	A drastic change in the mission makes the system obsolete or unnecessary.

For example, a computer security expert involved in risk assumption monitoring learns of a new attack technique that might threaten the security and survivability of the existing system. Let's assume that this new attack technique cannot be countered by straightforward maintenance activities such as applying a security patch to a system component or adding a new rule to a firewall. Based on the new attack technique, the security expert generates a set of attack scenarios to be used as input for a security and survivability analysis of the existing system. If deficiencies in the system's resistance to this new attack (or in the system's ability to recognize or recover from the attack) are discovered, then one or more life cycle activities, such as a modification of the system architecture or a change in security and survivability requirements, will be necessary.

The completion of one life cycle activity may trigger the need for another. Adjustments in the design tradeoffs with other system quality attributes may also be called for. For example, a specific architectural change meant to improve security may have unanticipated adverse effects on some other system quality attributes. These implicit tradeoffs can be systematically evaluated and explicitly adjusted using the results of an architecture tradeoff analysis [Kazman 98²⁶]. The point at which the evolutionary design process stops is dependent on the risk tolerance of the organization, and the perceived cost/benefit ratio, with respect to the particular set of trigger events. If evolution is not feasible, the organization may tolerate the risk or seek other alternatives that transcend the system.

It is essential that the evolutionary design activities take place in the context of full access to a comprehensive set of artifacts of the design process (such as descriptions of the rationale for tradeoffs made during the last design cycle). Continuity of at least the core members of the design team is particularly crucial for the evolutionary design of survivable systems so that the mission-specific design expertise can be sustained throughout the life of the system. Otherwise, the evolutionary design process will likely degenerate into patching, which can never support the long-term security of systems. Just as security must be designed into a system from the beginning and not tacked on later as an afterthought, long-term security cannot be sustained through patching or routine maintenance but only through the continual incorporation of new security and survivability solutions through a principled evolutionary design process. The development and promulgation of a suite of best practices to support this process would be a fundamental contribution to the software engineering profession.

The Need for Best Practices in Evolutionary Design

While evolutionary design is a critical aspect of building security in, there are few best practices for evolution that are supported by ample evidence and general consensus in the software engineering community. Those practices that do exist are typically classified under software maintenance.

Moreover, many aspects of evolutionary design are not yet well understood by the software engineering community. For example, it is not practical (i.e., not economically feasible) for a system to evolve along all

26. #dsy467-BSI_kazman

of the possible dimensions outlined in Table 1. How to decide during the initial design of a system which dimensions of change are most likely and how to make them amenable to low-cost redesign or automated upgrades is an important topic for further research and investigation. Some limited success in this area has been achieved through automatic upgrades of firewall rules and databases of attack signatures for detecting and eliminating viruses and other malware—essentially rapidly evolving a system in response to evolving security threats.

One of the most problematic aspects of the evolution of a secure and survivable system is how to recognize changes in security and survivability risk that arise as a result of your system's dependence on components, services, and systems (in a system-of-systems environment) whose design, implementation, and operation are not in your direct control. For example, changes in a vendor's development processes, underlying technology, personnel, ownership, or suppliers may increase security risks to your own system. Risks introduced or increased by changes in the elements that your system is dependent upon can be addressed in a variety of ways—often at the architectural level—but, of course, only if you are aware that external changes may have affected your earlier risk management assumptions.

An emerging area of research and engineering known as *assurance cases*²⁷ [Lipson 08²⁸] has great potential for helping system designers, integrators, architects, developers, acquisition personnel, and other stakeholders deal with the changing risks associated with the evolution of components, services, and systems. This would include a typical situation in which components are being integrated into a system by an organization that is not in control of the components' development or evolution. Assurance cases use *arguments* that refer to bodies of *evidence* to demonstrate that specific *claims* about a system (or component) are satisfied (e.g., with respect to a particular quality attribute). Assurance cases have been used successfully in the safety domain (where they are known as *safety cases*) and efforts are underway in the research community to extend the use of assurance cases to the security domain [Bloomfield 06a²⁹, Bloomfield 06b³⁰].

If the development of assurance cases for security were integrated into the system development life cycle as a best practice, then assurance cases would evolve along with the component or system being developed, and stakeholders could be notified of any change in an assurance case. One can envision an environment in which assurance cases would be the key to tracking the evolution of a system, and the elements that the system is dependent upon, with respect to the ongoing satisfaction of security claims. Assurance cases could then be used as a means of recognizing those evolutionary changes that impact the system's security and survivability risks. Research is needed not only to create and refine methods and tools for developing assurance cases for security but also to ensure that assurance cases can be developed in a cost-effective manner in order to provide practical support for evolutionary design.

In summary, further research and reduction to practice is urgently needed on a broad range of evolutionary design considerations for building secure and survivable systems, including evolutionary design principles and candidate best practices solicited from the software engineering community. The goal is to begin to formulate a set of demonstrably useful, highly actionable best security practices to support evolutionary design. Ensuring that you can effectively evolve the security capabilities of a system in response to a changing risk environment is an essential part of building security in—that is, building in the capability to evolve and improve the security and survivability of a system over its full lifetime of use.

References

- | | |
|------------------|--|
| [Bloomfield 06a] | Bloomfield, R. E.; Guerra, S.; Masera, M.; Miller, A.; Saydjari, O. S. <i>Assurance Cases for Security Workshop Report</i> ³¹ , Version 01c. Workshop on Assurance Cases for Security, June 13-15, 2005, Arlington, VA. |
|------------------|--|

27. <http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/assurance.html> (Assurance Cases)

28. #dsy467-BSI_lipson08

29. #dsy467-BSI_blooma

30. #dsy467-BSI_bloomb

- [Bloomfield 06b] Bloomfield, R. E.; Guerra, S.; Miller, A.; Masera, M.; & Weinstock, C. B. "International Working Group on Assurance Cases (for Security)." *IEEE Security & Privacy* 4, 3 (May-June 2006): 66-68.
- [Carney 05] Carney, David; Fisher, David; & Place, Patrick. *Topics in Interoperability: System-of-Systems Evolution*³² (CMU/SEI-2005-TN-002). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
- [ESA-CNES 96] European Space Agency (ESA) and National Center for Space Study (CNES) Inquiry Board (Prof. J. L. Lions, Chairman). *ARIANE 5 – Flight 501 Failure – Report by the Inquiry Board*³³. Paris: ESA and CNES, July 19, 1996.
- [Garlan 95] Garlan, David; Allen, Robert; & Ockerbloom, John. "Architectural Mismatch: Why Re-use Is So Hard." *IEEE Software* 12, 6 (November 1995): 17-26.
- [Kazman 98] Kazman, R.; Klein, M.; Barbacci, M.; Longstaff, T.; Lipson, H. F.; & Carriere, S. J. "The Architecture Tradeoff Analysis Method"³⁴. *Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 1998)*. Monterey, CA, USA, August 10-14, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.
- [Lehman 98] Lehman, M. M. "Software's Future: Managing Evolution." *IEEE Software* 15, 1 (January/February 1998): 40-44.
- [Lehman 01] Lehman, M. M. & Ramil, Juan F. "Rules and Tools for Software Evolution Planning and Management." In special issue on Software Management, *Annals of Software Engineering* 11, 1 (November 2001): 15-44.
- [Lipson 99] Lipson, Howard & Fisher, David. "Survivability—A New Technical and Business Perspective on Security"³⁵, 33–39. *Proceedings of the 1999 New Security Paradigms Workshop*. Caledon Hills, Ontario, Canada, Sept. 22–24, 1999. New York: Association for Computing Machinery, 2000.
- [Lipson 02] Lipson, H. F.; Mead, N.; & Moore, A. P. "Can We Ever Build Survivable Systems from COTS Components?" *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE' 02)*. Toronto, Ontario, Canada, May 27-31, 2002. Heidelberg, Germany: Springer-Verlag (LNCS 2348), 2002.

[Lipson 08]

Lipson, Howard. "Assurance Cases Overview³⁶." Build Security In Web site, January 2007 (Updated May 2008).

[Mead 01]

Mead, N. R.; Linger, R. C.; McHugh, J; & Lipson, H. F. "Managing Software Development for Survivable Systems." *Annals of Software Engineering* 11, 1 (November 2001): 45-78.

Carnegie Mellon Copyright

Copyright © Carnegie Mellon University 2005-2010.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu¹.

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

NO WARRANTY

THIS MATERIAL OF CARNEGIE MELLON UNIVERSITY AND ITS SOFTWARE ENGINEERING INSTITUTE IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

1. <mailto:permission@sei.cmu.edu>